Docket No. AUS9-2000-0561-US1

APPARATUS AND METHODS FOR SEQUENTIALLY SCHEDULING A PLURALITY OF COMMANDS IN A PROCESSING ENVIRONMENT WHICH EXECUTES COMMANDS CONCURRENTLY

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates to data processing systems, and further to a data processing system for scheduling, executing, and monitoring the execution of a plurality of commands in a data processing system which executes commands concurrently and which does not sense or test for the completion of execution of commands. More particularly, the present invention provides apparatus and methods for encapsulating each command in a process whose execution status is monitored so that the commands will execute sequentially wherein a first command will complete executing before beginning the executing of a next command.

2. Description of Related Art:

At the lowest level of interaction between computer operating systems and applications, certain fundamental differences arise regarding the handling of processes. In some environments, processes, such as scripts, programs, or commands, operate in a sequential manner such that execution of one process is not started until the previous process has completed executing. In these systems, the result from the execution of one process is, therefore, available when subsequent processes begin executing.

In other environments, multiple processes may start executing generally simultaneously, i.e. concurrently. In these

systems, the results of the execution of one process are not available to the other processes.

The environments which execute processes concurrently without regard for the status of execution of other processes suffer from some disadvantages. Branching logic and complex scripts cannot be constructed because the results of previously executed processes are not necessarily available to subsequent processes.

Thus, it would be beneficial to have an apparatus and method for scheduling and executing processes in a sequential order in an environment which executes processes concurrently.

SUMMARY OF THE INVENTION

A data processing system and method are disclosed for scheduling a sequential execution of multiple commands. The data processing system includes an environment which executes the commands concurrently. Without scheduling, each of the commands is executed in the environment without regard to a completion of execution of any other ones of the commands. Execution of the plurality of commands is scheduled in the environment so that the commands execute sequentially in programming order. When the commands are scheduled, a first one of the commands in the order begins and completes executing prior to a second one of the commands in the order beginning executing. When scheduled, the commands execute in the environment sequentially in programming order. In order to execute the commands sequentially, a process is spawned within which to execute the command. The execution status of the process is checked periodically by checking a process table. When the process has completed executing, i.e. when the command has completed executing, a new process is spawned within which to execute the next command in the sequential order.

Other features and advantages of the present invention will be described in, or will become apparent to those of ordinary skill in the art in view of the following detailed description of the preferred embodiments of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a data processing system according to the present invention;

Figure 2 is an exemplary block diagram of a data processing system according to the present invention;

Figure 3 is a flowchart which depicts the creation of a scheduler script to execute a plurality of commands sequentially in accordance with the present invention;

Figure 4 is a flow chart which illustrates the execution of a scheduler script in accordance with the present invention;

Figure 5 is a flow chart which illustrates determining whether a process is currently executing and setting a return code variable in accordance with the present invention;

Figure 6 depicts pseudo-code for a scheduler script in accordance with the present invention.; and

Figure 7 illustrates pseudo-code for a scheduler script including TSM commands inserted in the script in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of the present invention and its advantages are better understood by referring to Figures 1-7 of the drawings, like numerals being used for like and corresponding parts of the accompanying drawings.

The present invention is a method and system for scheduling a plurality of commands to be executed sequentially in an environment which executes commands concurrently. In the environment of the present invention, commands are executed generally simultaneously. The environment starts executing each command without regard for the completion of execution of any other commands.

The present invention provides a method and system for scheduling commands in this type of environment whereby a command will complete executing before a next command begins executing. In this manner, branching and complex scripts may be written which will execute properly in such an environment.

A plurality of commands are selected which are to be executed in a particular sequential order. Each command is inserted into a scheduler script in the sequential order. The scheduler script will control and monitor the execution of the commands. The scheduler script is then executed.

When the scheduler script is executed, the first command in the sequential order will be encapsulated within a process. This process will receive a process identifier which uniquely identifies this particular process. The process will begin executing. The command encapsulated within the process will then execute after the process has started executing. When the command has finished executing, the process will finish executing. When the process has finished executing, the next command in the scheduler script will be encapsulated in a new

process and will be executed in a manner similar to that described above.

The scheduler script will determine whether the process is executing by checking a return code variable. The return code variable is set equal to a first value to indicate that a process is executing and set equal to a second value to indicate that a process is not executing. The return code variable is set equal to the first value when the process starts executing. Thereafter, a process table is checked periodically for the process identifier which identifies the current process. The process identifier for each process which is currently executing will appear in the process table. When the process table indicates that the process is not executing, the return code variable will be set equal to the second value.

With reference now to the figures and in particular with reference to **Figure 1**, a pictorial representation of a data processing system in which the present invention may be implemented is depicted in accordance with a preferred embodiment of the present invention. A computer **100** is depicted which includes a system unit **110**, a video display terminal **102**, a keyboard **104**, storage devices **108**, which may include floppy drives and other types of permanent and removable storage media, and mouse **106**. Additional input devices may be included with personal computer **100**, such as, for example, a joystick, touchpad, touch screen, trackball, microphone, and the like. Computer **100** can be implemented using any suitable computer, such as an IBM RS/6000 computer or IntelliStation computer, which are products of International Business Machines Corporation, located in Armonk, New York. Although the depicted representation shows a computer, other embodiments of the present invention may be implemented in other types of data processing systems, such as a network computer. Computer **100** also preferably includes a

graphical user interface that may be implemented by means of systems software residing in computer readable media in operation within computer **100**.

With reference now to **Figure 2**, a block diagram of a data processing system is shown in which the present invention may be implemented. Data processing system **200** is an example of a computer, such as computer **100** in **Figure 1**, in which code or instructions implementing the processes of the present invention may be located. Data processing system **200** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processor **202** and main memory **204** are connected to PCI local bus **206** through PCI bridge **208**. PCI bridge **208** also may include an integrated memory controller and cache memory for processor **202**. Additional connections to PCI local bus **206** may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter **210**, small computer system interface SCSI host bus adapter **212**, and expansion bus interface **214** are connected to PCI local bus **206** by direct component connection. In contrast, audio adapter **216**, graphics adapter **218**, and audio/video adapter **219** are connected to PCI local bus **206** by add-in boards inserted into expansion slots. Expansion bus interface **214** provides a connection for a keyboard and mouse adapter **220**, modem **222**, and additional memory **224**. SCSI host bus adapter **212** provides a connection for hard disk drive **226**, tape drive **228**, and CD-ROM drive **230**. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

An operating system runs on processor **202** and is used to

coordinate and provide control of various components within data processing system **200** in **Figure 2**. The operating system may be a commercially available operating system such as the AIX operating system available from International Business Machines. The AIX operating system is a UNIX-type operating system. In addition, other operating systems such as Windows 2000, available from Microsoft Corporation, or object oriented systems such as Java may run on data processing system **200**. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on data processing system **200**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented programming system, and applications or programs are located on storage devices, such as hard disk drive **226**, and may be loaded into main memory **204** for execution by processor **202**.

Data processing system **200** includes an environment which executes commands concurrently. For example, data processing system **200** may include a Tivoli™ Storage System Manager (TSM) server system. TSM is a network backup and recovery tool that spans different manufacturers' platforms. TSM does not sense when a command has completed executing, and it does not test to determine when a command has completed executing. TSM executes commands concurrently. A process is spun off for each command generally simultaneously without regard for whether any others of the commands have finished executing.

Those of ordinary skill in the art will appreciate that the hardware in **Figure 2** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash ROM (or equivalent nonvolatile memory) or optical disk drives and the like, may be used in addition to or in place of the hardware

depicted in **Figure 2**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

For example, data processing system **200**, if optionally configured as a network computer, may not include SCSI host bus adapter **212**, hard disk drive **226**, tape drive **228**, and CD-ROM **230**, as noted by dotted line **232** in **Figure 2** denoting optional inclusion. In that case, the computer, to be properly called a client computer, must include some type of network communication interface, such as LAN adapter **210**, modem **222**, or the like. As another example, data processing system **200** may be a stand-alone system configured to be bootable without relying on some type of network communication interface, whether or not data processing system **200** comprises some type of network communication interface. As a further example, data processing system **200** may be a personal digital assistant (PDA), which is configured with ROM and/or flash ROM to provide non-volatile memory for storing operating system files and/or user-generated data.

The depicted example in **Figure 2** and above-described examples are not meant to imply architectural limitations. For example, data processing system **200** also may be a notebook computer or hand held computer in addition to taking the form of a PDA. Data processing system **200** also may be a kiosk or a Web appliance.

The processes of the present invention are performed by processor **202** using computer implemented instructions, which may be located in a memory such as, for example, main memory **204**, memory **224**, or in one or more peripheral devices **226-230**.

Figure 3 a flowchart which depicts the creation of a scheduler script to execute a plurality of commands sequentially in accordance with the present invention. The process starts as

depicted by block **300** and thereafter passes to block **302** which illustrates selecting a plurality of commands to be executed sequentially. Next, block **304** depicts determining a sequential order to use to execute the commands. Typically, the sequential order will be the programming order. Thereafter, block **306** illustrates inserting the commands into the scheduler script in the sequential order. The process then terminates as illustrated by block **308**.

Figure 4 is a flow chart which illustrates the execution of a scheduler script in accordance with the present invention. The process starts as depicted by block **400** and thereafter passes to block **402** which illustrates encapsulating a first command in a process that is identified by a process identifier. Next, block **404** depicts setting a return code variable for the process equal to a first value. The first value indicates that the process is currently executing. Thereafter, block **406** illustrates starting the execution of the process. Next, block **408** depicts starting the execution of the first command in response to starting the execution of the process. The command is executed within the process.

The process then passes to block **410** which illustrates a determination of whether or not the return code variable is equal to the second value. If a determination is made that the return code variable is not equal to the second value, the process passes back to block **410**. Referring again to block **410**, if a determination is made that the return code variable is equal to the second value, the process passes to block **412** which depicts a determination of whether or not this was the last command to execute. If a determination is made that this was not the last command to execute, the process passes to block **414** which illustrates getting a next command in the sequential order. The

next command in the sequential order will be the next command in the scheduler script. The commands were originally inserted into the scheduler script in the sequential order. Thereafter, block **416** depicts encapsulating the next command in the sequential order in a process. This process is identified by its own, unique process identifier. Next, block **418** illustrates setting the return code variable equal to the first value for this process. The process then passes back to block **410**.

Referring again to block **412**, if a determination is made that this was the last command to execute, the process passes to block **420**. Block **420** depicts returning control to the shell script which called this scheduler script.

Figure 5 is a high level flow chart which depicts a determination of whether a process is currently executing and setting a return code variable in accordance with the present invention. The process starts as illustrated by block **500** and thereafter passes to block **502** which depicts determining the process identifier which identifies the current process spawned to execute a command. Next, block **504** illustrates searching a process table for this process identifier. Block **506**, then, depicts a determination of whether or not the process table indicates that the process is currently running. While a process is running, the process identifier which identifies that process will appear in the process table. When the process has completed executing, the process identifier will be removed from the table and will no longer appear there.

Referring again to block **506**, if a determination is made that the process table indicates that the process is not running, the process passes to block **508** which illustrates setting the return code variable equal to a second value. When the return code variable is set equal to the second value, the return code

variable indicates that the process is not currently running. The process then terminates as depicted by block **510**.

Referring again to block **506**, if a determination is made that the process table indicates that the process is running, i.e. the process identifier for this process appears in the process table, the process passes to block **512**. Block **512** illustrates setting a timer. Next, block **514** depicts a determination of whether or not the timer has expired. If a determination is made that the timer has not expired, the process loops back to block **514** until a determination is made that the timer has expired. Referring again to block **514**, if a determination is made that the timer has expired, the process passes back to block **504** in order to again search the process table for the process identifier. In this manner, the process table is checked periodically to determine whether the process identifier appears in the table. The periods are determined by the length of time set using the timer.

Figure 6 depicts pseudo-code for a scheduler script in accordance with the present invention. As depicted, a first command may be inserted into a first process. A second command may be inserted into a second process. And, a third command may be inserted into a third process. When the scheduler script is executed, the first process and first will be executed. When the first process is completed, the second process and second command will be executed. And, when the second process is complete, the third process and third command will be executed.

Figure 7 illustrates pseudo-code for a scheduler script including TSM commands inserted in the script in accordance with the present invention.

In this manner, commands may be executed sequentially in an environment which executes commands concurrently. A scheduler script is provided which encapsulates each command in a process.

The scheduler script then executes the process and monitors the current execution status of the process by monitoring a return code variable. The return code variable is set equal to a first value to indicate that a process is currently executing. A process table is checked to determine whether a process is currently executing. The process identifier for a process will appear in the process table when the process is currently executing. When the process identifier no longer appears in the process table, the return code variable is set equal to a second value to indicate that the process has completed executing. When the return code variable is set equal to the second value, the scheduler will get the next command and encapsulate it within a process. In this manner, the commands are executed sequentially where each command completes executing prior to commencing the execution of the next command in the sequential order.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, CD-ROMs, and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen

and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.